	ATL Transformation Example	Author Éric Vépa evepa@sodius.com
	UML2 to Measure	August 30th , 2007

1. ATL Transformation Example: UML2 to Measure

The UML2 to Measure example describes measurement on UML2 models, using metrics defined as ATL libraries.

1.1. Transformation Overview

The aim of this transformation is to collect measurement data on UML2 meta-models.

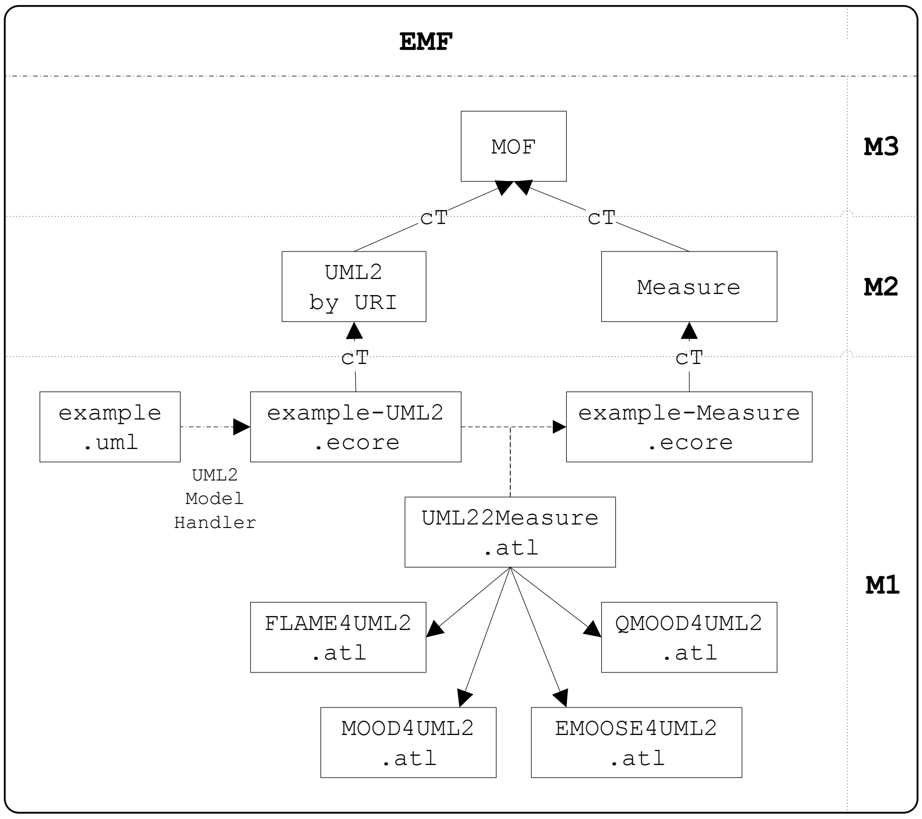

Figure 1: Overview of the transformation

UML2 models can be measured with ATL transformations. An UML2 model file is loaded using the UML2 model handler, the result model is conformed to the UML2 Eclipse Project [3] and used as the input of the transformation. The transformation input and output meta-model handlers are UML2 and Measure. The run of the transformation *UML22Measure* produces a collection of measurement data.

We obtain an output model of measures (which keeps the hierarchy of the model). The metrics used in the transformation are implemented with ATL libraries and will be explained in an upcoming section.

2. Meta-models

2.1. UML2

The UML2 meta-model used is from the UML2 Eclipse Project [3].

2.2. Measure

The Measure meta-model is used to stored the data collected after a model measurement.

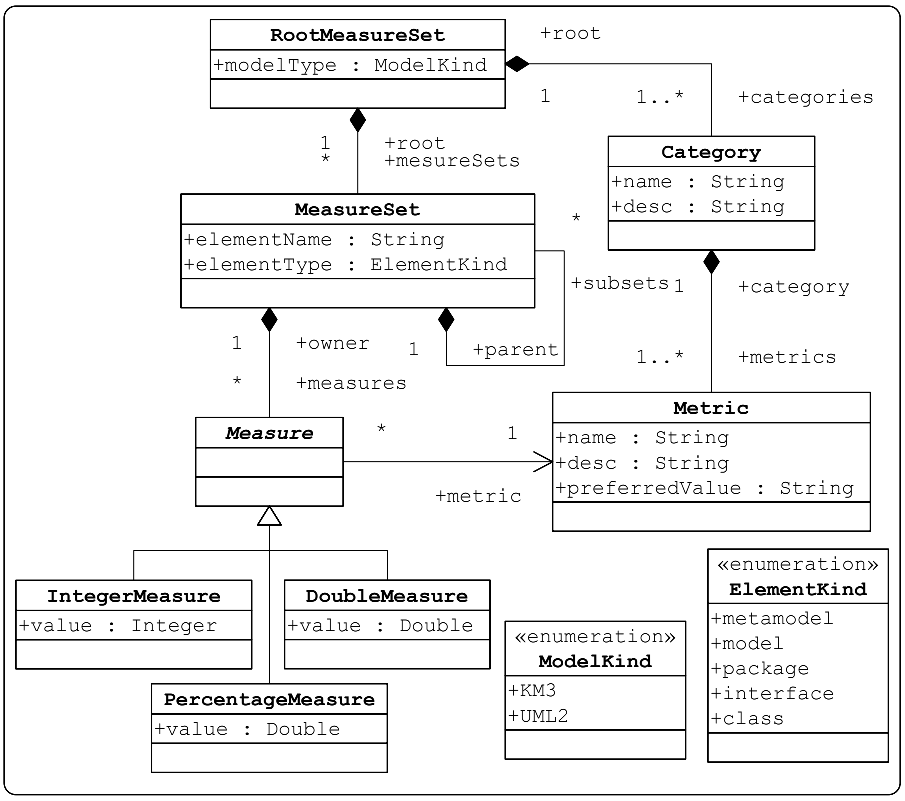

Figure 2: Measure meta-model

A measure model is in the following way made up: the root is a set of measure (*RootMeasureSet*) which contains information on the type of measured model (*modelType* among *KM3* or *UML2*), a set of categories of metric and sets of measure for each model element measured. A category (*Category*) corresponds to a metric set with a *name* and a description (*desc*) (an acronym and its definition). A category gathers one or more metric (*Metric*) also defined with a *name* and a description. A default predicate is also associated (*preferredValue*), it is the desired values for the metric (for example ≠ 0 or > 75). A set of measure (*MeasureSet*) described measurements performed on a model element (*elementName*) of a given type (*elementType* among *meta-model*, *model*, *package*, *interface* or *class*). The set of measure are structured between them, for example a set of measure on a package will contain the set of measure of the classes that this package contains. A measure (*Measure*) is associated to a metric and is declined in several versions. Measures with an integer, real or percentage value (respectively *IntegerMesasure*, *DoubleMeasure* and *PercentageMeasure*).

3. Transformation from UML2 to Measure

3.1. Rules specification

These are the rules to collect measurement data from a UML2 model to a Measure model.

- For the whole model, the following elements are created:
  - A *RootMeasureSet* element is created with:
    - A type of model measured (*modelType* set to *#UML2*).
  - For each category implemented, the following elements are created:
    - A *Category* element with :
      - A name and a description.
      - The created *Category* element is linked to the *RootMeasureSet*.
  - For each metric implemented for a category, the following elements are created:
    - A *Metric* element with:
      - A name and a description.
      - A mandatory preferred value.
      - The created *Metric* element is linked to a *Category* element.

  The measure level determinates the metrics and categories that are registered.
- For each *Package* element, the following elements are created:
  - A *MeasureSet* element with the name and the type of the *Package* element measured.
  - The created *MeasureSet* element is linked to the *MeasureSet* created for his owner *Package* element.

  - If the *Package* element contains *Class* elements, the following elements are created:
    - An *IntegerMeasure*, *DoubleMeasure* or *PercentageMeasure* element, for each *Metric* element created and defined for package level.
  - If the *Package* element is a root package :
    - It is linked to the *RootMeasureSet*.
- For each *Class* element, the following elements are created:
  - A *MeasureSet* element with the name and the type of the *Class* element measured.
  - The created *MeasureSet* element is linked to the *MeasureSet* created for his owner *Package* or *Class* (nested classifier) element.
  - An *IntegerMeasure*, *DoubleMeasure* or *PercentageMeasure* element, for each *Metric* element created and defined for class level.

3.2. ATL code

This ATL code for the *UML22Measure* transformation consists in 4 helpers and 9 rules.

The transformation uses the metrics libraries defined in section 4.

The attribute helper *measureLevel* is used to define the type of model elements measured. For example, at package level (*#package*), only metrics defined for packages will be used. At class level (*#class*), both packages and classes metrics will be used.

The two maps *CategoryByName* and *MetricByName* are used to register the categories of metrics and the metrics implemented.

The entrypoint rule *Metrics* is used to fill the two previous maps, before processing measures. The metrics and categories registered depend on the measure level.

The rule *Package2MeasureSet* is called if the package or class level is enabled. If the package contains some classes, measures will be performed for the metrics defined for package level.

The rule *Class2MeasureSet* is called if the class level is enabled. Measures are performed for each metrics defined for class level.

The called rules *Category*, *Metric* and *MetricWithPreferredValue* are used in the entrypoint rule to register the implemented categories and metrics with mandatory preferred value.

The called rules *IntegerMeasure*, *DoubleMeasure* and *PercentageMeasure* store the value for a metric given.

4. Metrics Libraries

4.1. FLAME for UML2 Library

4.1.1. FLAME (Formal Library for Aiding Metrics Extraction)

The functions of this library are defined in OCL language in [4] and [5] for the UML 1.3 meta-model. They have been adapted to fit with the UML2 meta-model and class diagram models.

4.1.2. ATL code

This ATL code for the *FLAME4UML2* library consists in 93 helpers.

4.2. MOOD for UML2 Library

4.2.1. MOOD (Metrics for Object-Oriented Design) and MOOD2

Name	**MOOD::AIF** - Attributes Inheritance Factor
Informal definition	Quotient between the number of inherited attributes in all classes of the package and the number of available attributes (locally defined plus inherited) for all classes of the current package.
Name	**MOOD::OIF** - Operations Inheritance Factor
Informal definition	Quotient between the number of inherited operations in all classes of the package and the number of available operations (locally defined plus inherited) for all classes of the current package.
Name	**MOOD::AHF** - Attributes Hiding Factor
Informal definition	Quotient between the sum of the invisibilities of all attributes defined in all classes in the current package and the total number of attributes defined in the package.
Name	**MOOD::OHF** - Operations Hiding Factor
Informal definition	Quotient between the sum of the invisibilities of all operations defined in all classes in the current package and the total number of operations defined in the package.
Name	**MOOD::BPF** - Behavioral Polymorphism Factor
Informal definition	Quotient between the actual number of possible different polymorphic situations within the current package and the maximum number of possible distinct polymorphic situations (due to inheritance).
Name	**MOOD::CCF** - Class Coupling Factor
Informal definition	Quotient between the actual number of coupled class-pairs within the package and the maximum possible number of class-pair couplings in the package. This coupling is the one not imputable to inheritance.

Name	**MOOD::ICF** - Internal Coupling Factor
Informal definition	Quotient between the number of coupling links where both the client and supplier classes belong to the current package and the total number of coupling links originating in the current package.
Name	**MOOD2::IIF** - Internal Inheritance Factor
Informal definition	Quotient between the number of inheritance links where both the base and derived classes belong to the current package and the total number of inheritance links originating in the current package.
Name	**MOOD2::AHEF** - Attributes Hiding Effectiveness Factor
Informal definition	Quotient between the cumulative number of the package classes that do access the package attributes and the cumulative number of the package classes that can access the package attributes.
Name	**MOOD2::OHEF** - Operations Hiding Effectiveness Factor
Informal definition	Quotient between the cumulative number of the package classes that do access the package operations and the cumulative number of the package classes that can access the package operations.

4.2.2. ATL code

This ATL code for the *MOOD4UML2* library consists in 10 helpers.

The implemented metrics from the MOOD and MOOD2 sets only depend on the FLAME functions and are list above. These metrics are defined for package level.

4.3. EMOOSE for UML2 Library

4.3.1. MOOSE (Metrics for Object-Oriented Software Engineering) and EMOOSE (Extended MOOSE)

Name	**MOOSE::DIT** - Depth of Inheritance Tree
Informal definition	The length of the longest path of inheritance from the current class to the root of the tree.
Name	**MOOSE::NOC** - Number Of Children
Informal definition	The number of classes that inherit directly from the current class.
Name	**MOOSE::CBO** - Coupling Between Objects
Informal definition	The number of other classes that are coupled to the current one. Two classes are coupled when references declared in one class use references or instance variables defined by the other class. Or used as a type or in reference by other classes.
Name	**MOOSE::RFC** - Response for a Class
Informal definition	The number of methods in the current class that might respond to a message received by its object, including methods both inside and outside of this class.

Name	**EMOOSE::SIZE2**
Informal definition	Number of local attributes and operations defined in the class. The metric SIZE 1 is code dependant so not adapted to our problem.

4.3.2. ATL code

This ATL code for the *EMOOSE4UML2* library consists in 6 helpers.

The implemented metrics from the MOOSE and EMOOSE sets only depend on the FLAME functions and are list above. These metrics are defined for class level.

4.4. QMOOD for UML2 Library

4.4.1. QMOOD (Quality Model for Object-Oriented Design)

Name	**QMOOD::DSC** - Design Size in Classes
Informal definition	Count of the total number of classes in the design.
Name	**QMOOD::NOH** - Number of Hierarchies
Informal definition	Count of the number of class hierarchies in the design.
Name	**QMOOD::NIC** - Number of Independent Classes
Informal definition	Count of the number of Classes that are not inherited by any Class in the design.
Name	**QMOOD::NSI** - Number of Single Inheritance
Informal definition	Number of Classes (sub classes) that use inheritance in the design.
Name	**QMOOD::NNC** - Number of Internal Classes
Informal definition	Count of the number of internal classes defined for creating generalization-specialization structures in class hierarchies of the design.
Name	**QMOOD::NAC** - Number of Abstract Classes
Informal definition	Count of the number of classes that have been defined purely for organizing information in the design.
Name	**QMOOD::NLC** - Number of Leaf Classes
Informal definition	Count of the number of leaf classes in the hierarchies of the design.
Name	**QMOOD::ADI** - Average Depth of Inheritance
Informal definition	The average depth of inheritance of classes in the design. It is computed by dividing the summation of maximum path lengths to all classes by the number of classes. The path length for a class is the number of edges from the root to the class in an inheritance tree representation.
Name	**QMOOD::AWI** - Average Width of Inheritance
Informal definition	The average number of children per class in the design. The metric is computed by dividing the summation of the number of children over all classes by the number of classes in the design

Name	**QMOOD::ANA** - Average Number of Ancestors
Informal definition	The average number of classes from which a class inherits information.
Name	**QMOOD::MFA** - Measure of Functional Abstraction
Informal definition	The ratio of the number of methods inherited by a class to the total number of methods accessible by members in the class.
Name	**QMOOD::MAA** - Measure of Attribute Abstraction
Informal definition	The ratio of the number of attributes inherited by a class to the total number of attributes in the class.
Name	**QMOOD::MAT** - Measure of Abstraction
Informal definition	The average of functional and attribute abstraction measures.
Name	**QMOOD::MOA** - Measure of Aggregation
Informal definition	Count of the number of data declarations whose types are user defined classes.
Name	**QMOOD::MRM** - Modeled Relationship Measure
Informal definition	Measure of the total number of attribute and parameter based relationships in a class.
Name	**QMOOD::DAM** - Data Access Metric
Informal definition	The ratio of the number of private attributes to the total number of attributes declared in a class.
Name	**QMOOD::OAM** - Operation Access Metric
Informal definition	The ratio of the number of public methods to the total number of methods declared in the class.
Name	**QMOOD::MAM** - Member Access Metric
Informal definition	This metric computes the access to all the members (attributes and methods) of a class.
Name	**QMOOD::NOA** - Number of Ancestors
Informal definition	Counts the number of distinct classes which a class inherits.
Name	**QMOOD::NOM** - Number of Methods
Informal definition	Count of all the methods defined in a class.
Name	**QMOOD::CIS** - Class Interface Size
Informal definition	Number of public methods in a class.
Name	**QMOOD::NPT** - Number of Unique Parameter Types
Informal definition	Number of different parameter types used in the methods of the class.
Name	**QMOOD::NPM** - Number of Parameters per Method
Informal definition	Average of the number of parameters per method in the class. Computed by summing the parameters of all methods and dividing by the number of methods in the class.
Name	**QMOOD::NOD** - Number of Attributes
Informal definition	Number of attributes in the class.
Name	**QMOOD::NAD** - Number of Abstract Data Types
Informal definition	Number of user defined objects used as attributes in the class and which are necessary to instantiate an object instance of the (aggregate) class.

Name	**QMOOD::NPA** - Number of Public Attributes
Informal definition	Number of attributes that are declared as public in the class.
Name	**QMOOD::CSM** - Class Size Metric
Informal definition	Sum of the number of methods and attributes in the class.
Name	**QMOOD::CAM** - Cohesion Among Methods of Class
Informal definition	Computes the relatedness among methods of the class based upon the parameter list of the methods. The metrics is computed using the summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the class.
Name	**QMOOD::DCC** - Direct Class Coupling
Informal definition	Count of the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in methods.
Name	**QMOOD::MCC** - Maximum Class Coupling
Informal definition	This metric not only includes classes that are directly related to a class by attributes and methods, but also classes that are indirectly related through the directly related classes.
Name	**QMOOD::DAC** - Direct Attribute Base Coupling
Informal definition	This metric is a direct count of the number of different class types that are declared as attribute references inside a class.
Name	**QMOOD::DPC** - Direct Parameter Based Coupling
Informal definition	Number of class object types that are required directly for a message passing (parameters) to methods in the class.
Name	**QMOOD::MPC** - Maximum Parameter Based Coupling
Informal definition	Number of Class object types that are required directly and indirectly for message passing (parameters) in the Class.
Name	**QMOOD::CCD** - Class Complexity Based on Data
Informal definition	Computes complexity based upon the number of components (attributes) that are defined in the class. All component declarations are resolved to the basic primitives (integers, doubles and characters). The metric value is a count of the number of primitives.
Name	**QMOOD::CCP** - Class Complexity Based on Method Parameters
Informal definition	Estimates complexity based upon the number of parameters required to call methods of the Class. Inherited method parameters are also included in the computation of the metric value.
Name	**QMOOD::CCM** - Class Complexity Based on Members
Informal definition	This metric is an aggregate of the data and method parameter complexities.

4.4.2. ATL code

This ATL code for the *QMOOD4UML2* library consists in 36 helpers.

The implemented metrics from the QMOOD set only depends on the FLAME functions and are list above. These metrics are defined both for package and class levels.

5. References

[1] ATLAS (ATLantic dAta Systems) Official Webpage: http://www.sciences.univ-nantes.fr/lina/ATLAS/

[2] AM3 ANT Tasks: http://wiki.eclipse.org/index.php/AM3_Ant_Tasks

[3] UML2 Project Official Webpage: http://www.eclipse.org/modeling/mdt/?project=uml2

[4] Baroni, A.L.: *Formal Definition of Object-Oriented Design Metrics*. Master Thesis, Vrije University, Brussel, Belgium, 2002.

[5] Baroni, A.L. and Abreu, F.B.: *A Formal Library for Aiding Metrics Extraction*. In: Workshop on Object-Oriented Reengineering (ECOOP'03), Darmstadt, Germany, July 2003.